ELECTRICAL
& COMPUTER
ENGINEERING

ECE-595 Network Softwarization

PROF. FABRIZIO GRANELLI (FABRIZIO.GRANELLI@UNITN.IT)

PROF. MICHAEL DEVETSIKIOTIS (MDEVETS@UNM.EDU)

Warning – Hands-on Software

Following hands-on activities will be performed using ComNetsEmu VM

ComNetsEmu network emulator includes:

- Mininet
- Docker
- Several SDN/NFV examples

Instructions for installation here:

https://git.comnets.net/public-repo/comnetsemu

SDN and the Planes of Networking

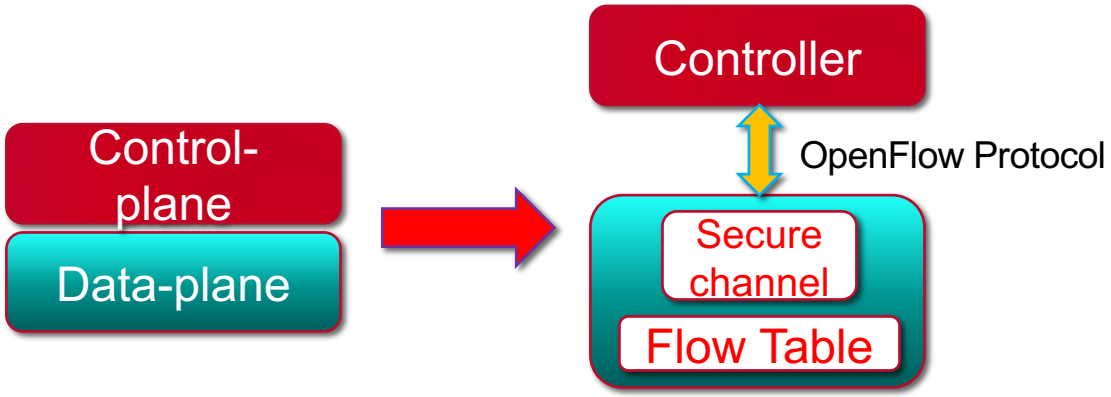
Data Plane: All activities involving as well as resulting from data packets sent by the end user, e.g.,

- Forwarding
- Fragmentation and reassembly
- Replication for multicasting

Control Plane: All activities that are necessary to perform data plane activities but do not involve end-user data packets

- Making routing tables
- Setting packet handling policies (e.g., security)

Separation of Control and Data Plane



Control logic is moved to a central controller

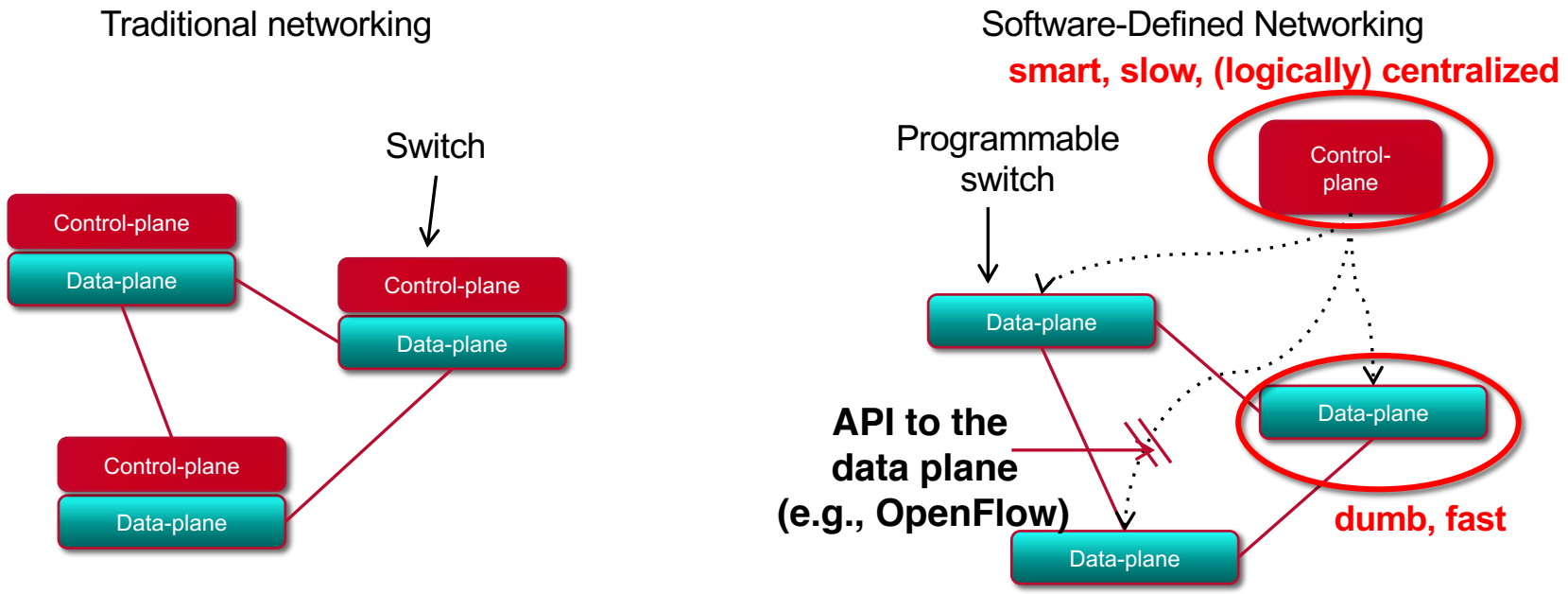
Switches only have forwarding elements

One expensive controller with a lot of cheap switches

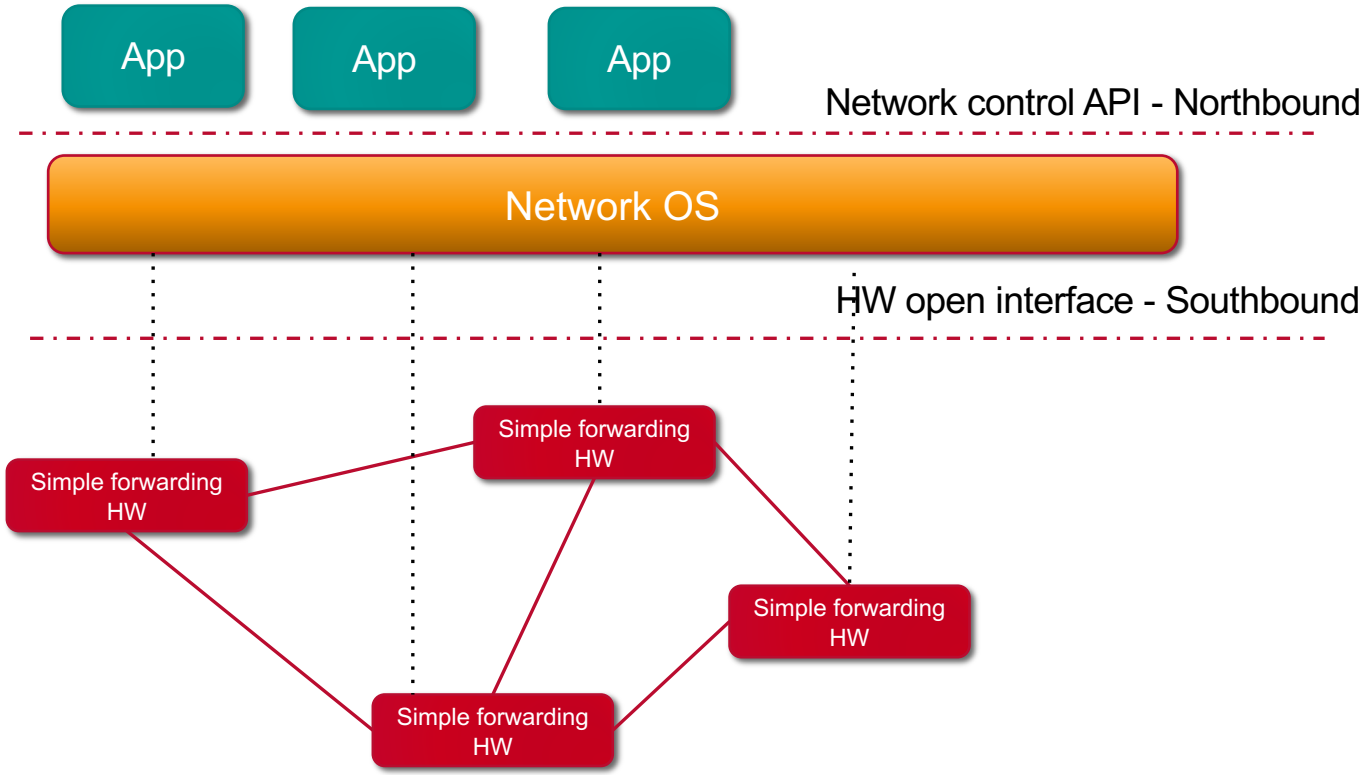
OpenFlow is the protocol to send/receive forwarding rules from controller to switches

By programming the controller, we can quickly change the entire network behavior → Software Defined Networking

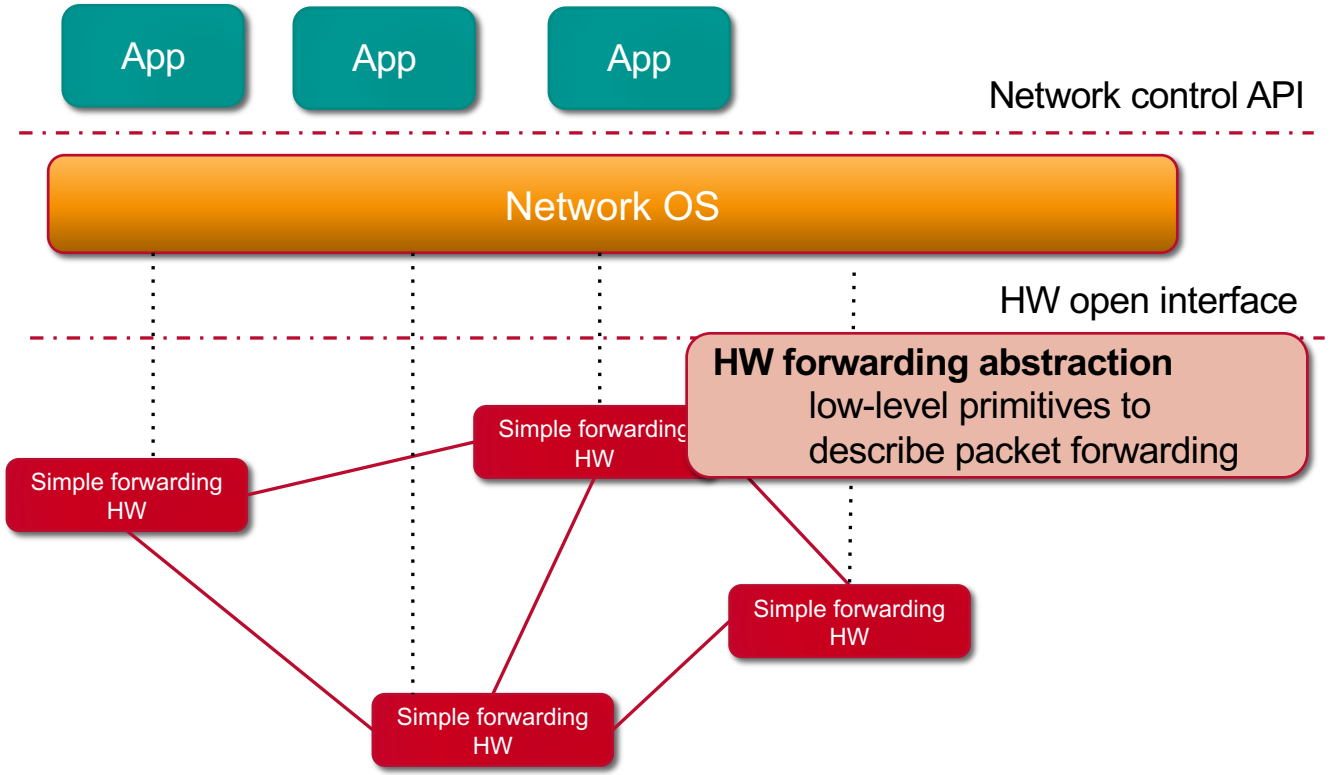
The SDN paradigm



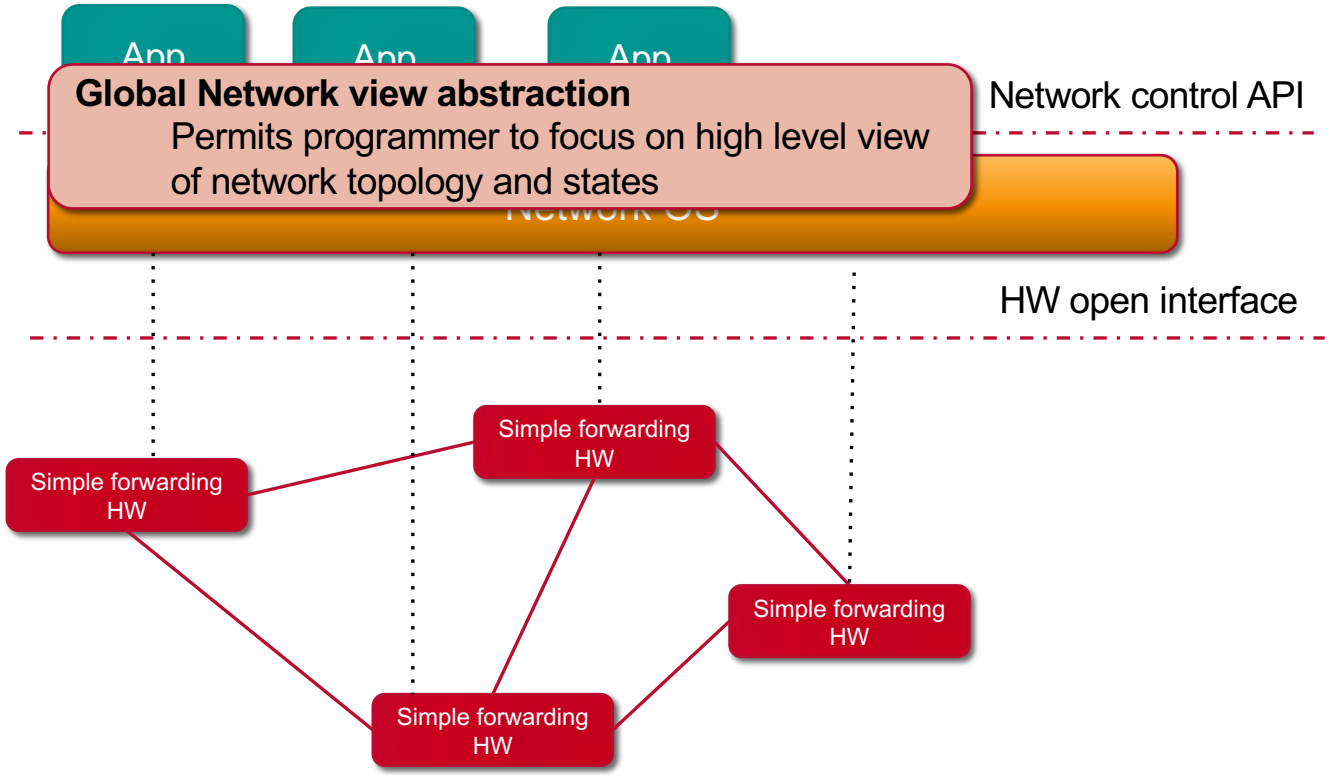
SDN architecture, sketch



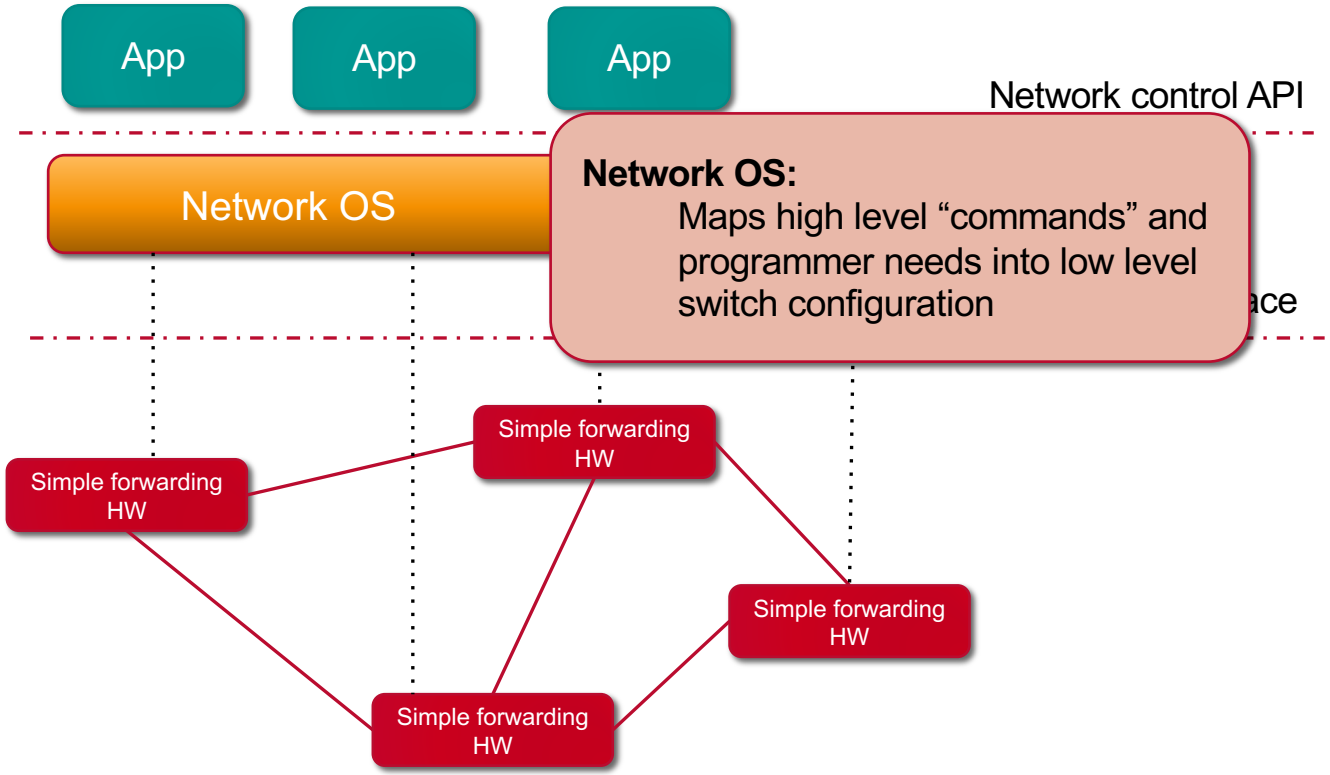
SDN architecture, sketch



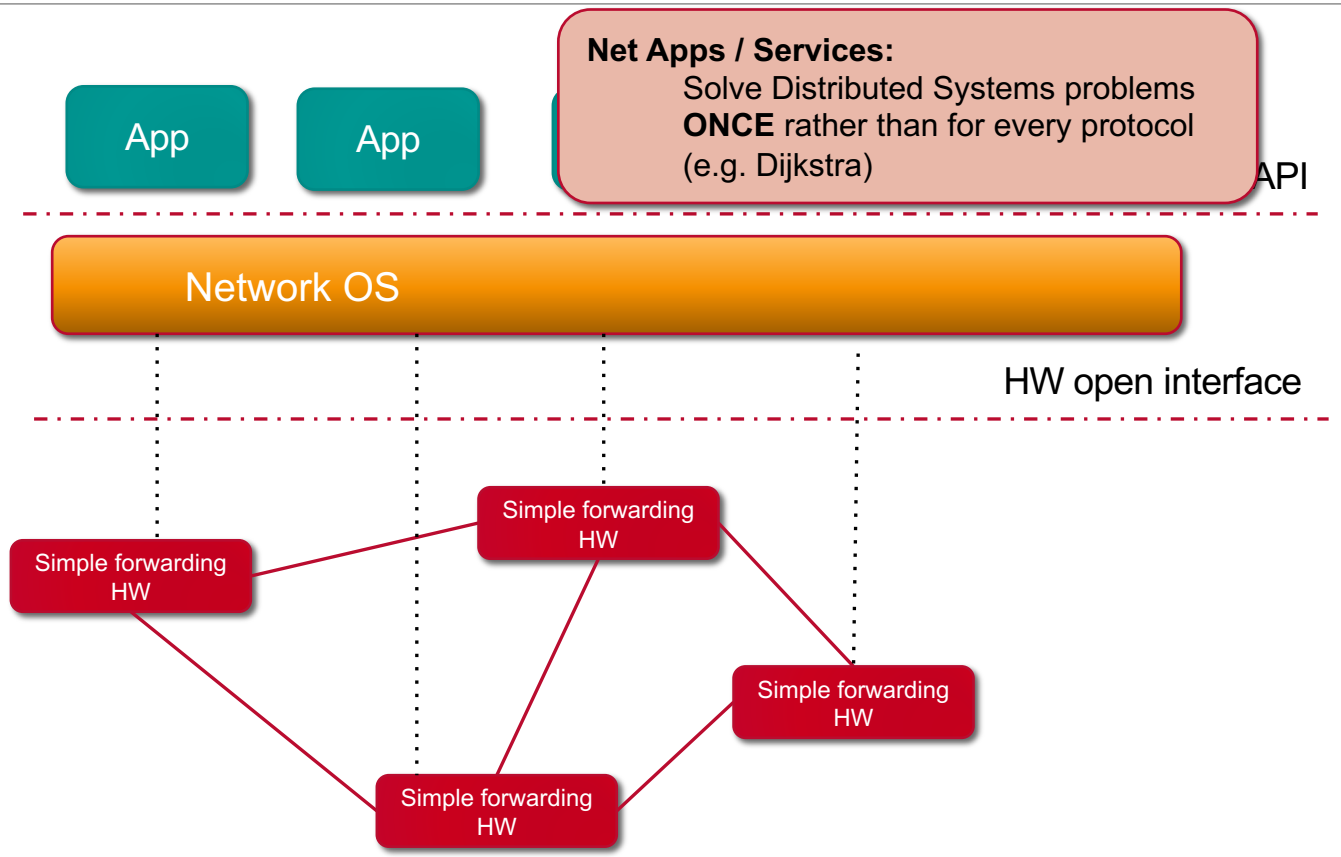
SDN architecture, sketch



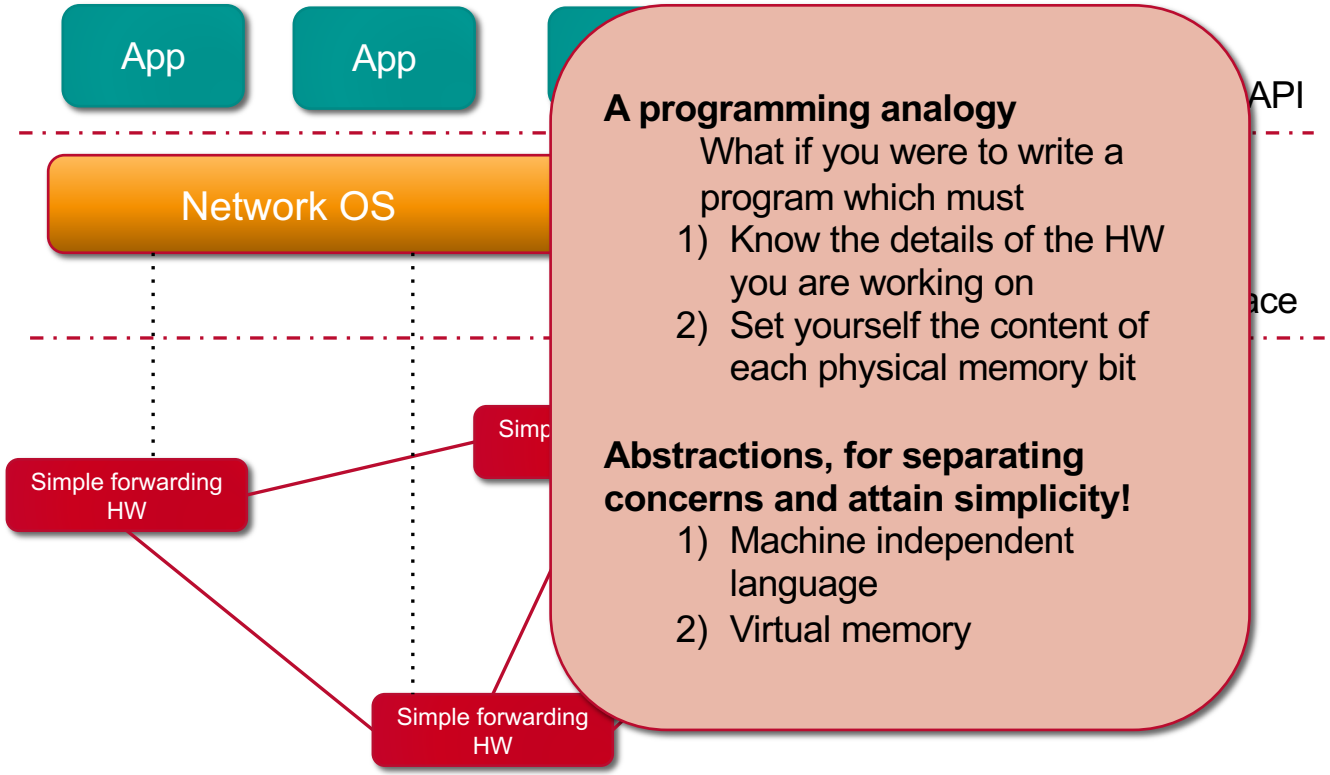
SDN architecture, sketch



SDN architecture, sketch



SDN architecture, sketch



What is SDN? [ONF Definition]

"The physical separation of the network control plane from the forwarding plane, and where a control plane controls several devices."

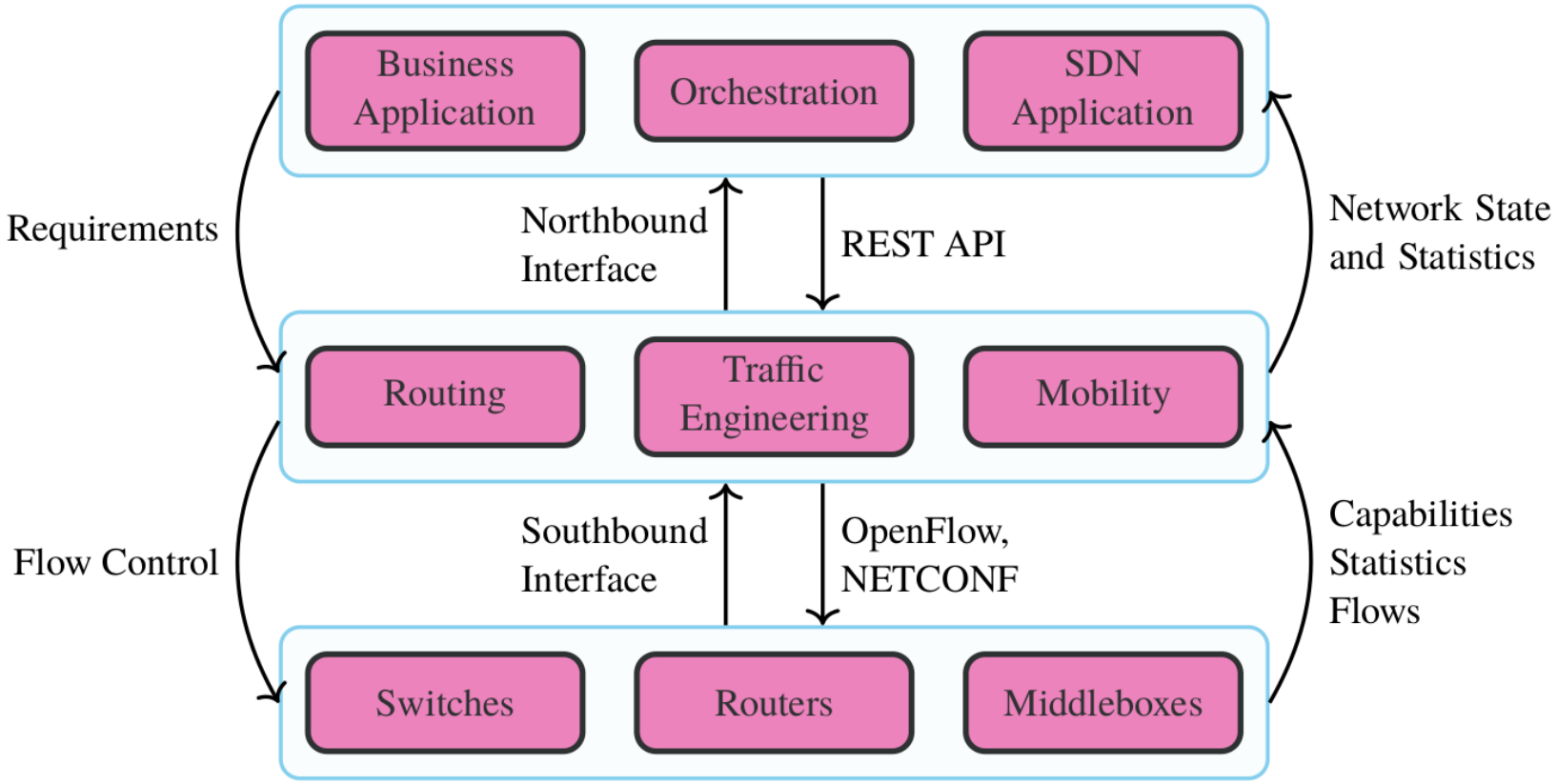
1. Directly programmable
2. Agile: Abstracting control from forwarding
3. Centrally managed
4. Programmatically configured
5. Open standards-based vendor neutral

https://www.opennetworking.org/index.php?option=com_content&view=article&id=686&Itemid=272&lang=en

What do we need SDN for?

1. Virtualization: Use network resource without worrying about where it is physically located, how much it is, how it is organized, etc.
2. Orchestration: Manage thousands of devices
3. Programmable: Should be able to change behavior on the fly.
4. Dynamic Scaling: Should be able to change size, quantity
5. Automation: Lower OpEx
6. Visibility: Monitor resources, connectivity
7. Performance: Optimize network device utilization
8. Multi-tenancy: Sharing expensive infrastructure
9. Service Integration
10. Openness: Full choice of Modular plug-ins
11. Unified management of computing, networking, and storage

SDN architecture overview



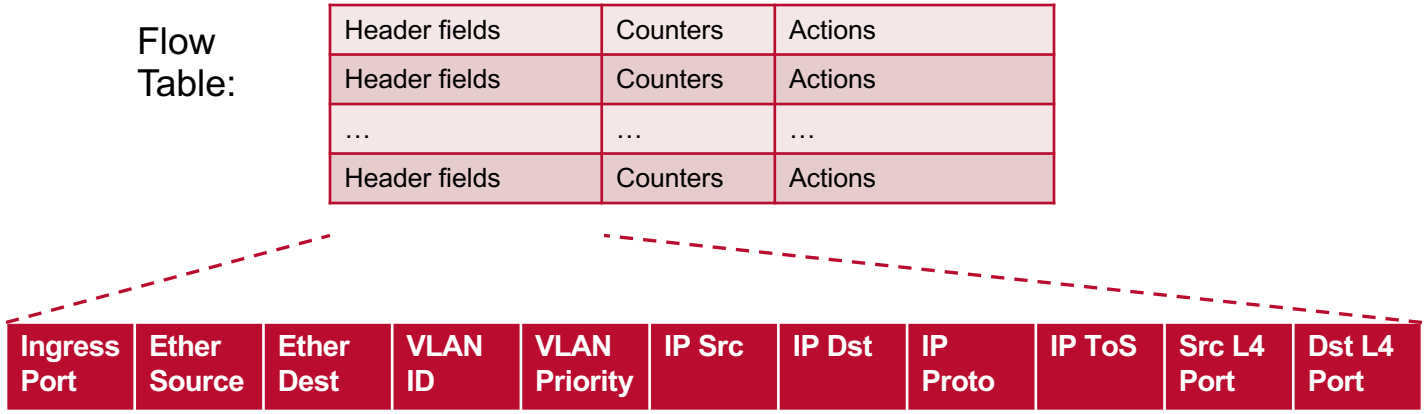
Technologies and Standards

Several options are available, while the SDN concept remain the same

- OpenFlow
- P4
- NETCONF

OpenFlow V1.0

On packet arrival, match the header fields with flow entries in a table; if any entry matches, update the counters indicated in that entry and perform indicated actions, otherwise contact controller



Flow Table example

Port	Src MAC	Dst MAC	VLAN ID	Prio	Ether Type	Src IP	Dst IP	IP Proto	IP ToS	Src L4 Port	Dst L4 Port	Action	Counter
*	*	0A:C8:*	*	*	*	*	*	*	*	*	*	Port 1	102
*	*	*	*	*	*	*	192.168.*.*	*	*	*	*	Port 2	202
*	*	*	*	*	*	*	*	*	*	21	21	Drop	420
*	*	*	*	*	*	*	*	0x806	*	*	*	Local	444
*	*	*	*	*	*	*	*	0x1*	*	*	*	Controller	1

Idle timeout: Remove entry if no packets received for this time

Hard timeout: Remove entry after this time

If both are set, the entry is removed if either one expires.

Actions

Forward to Physical Port *i* or to *Virtual Port*:

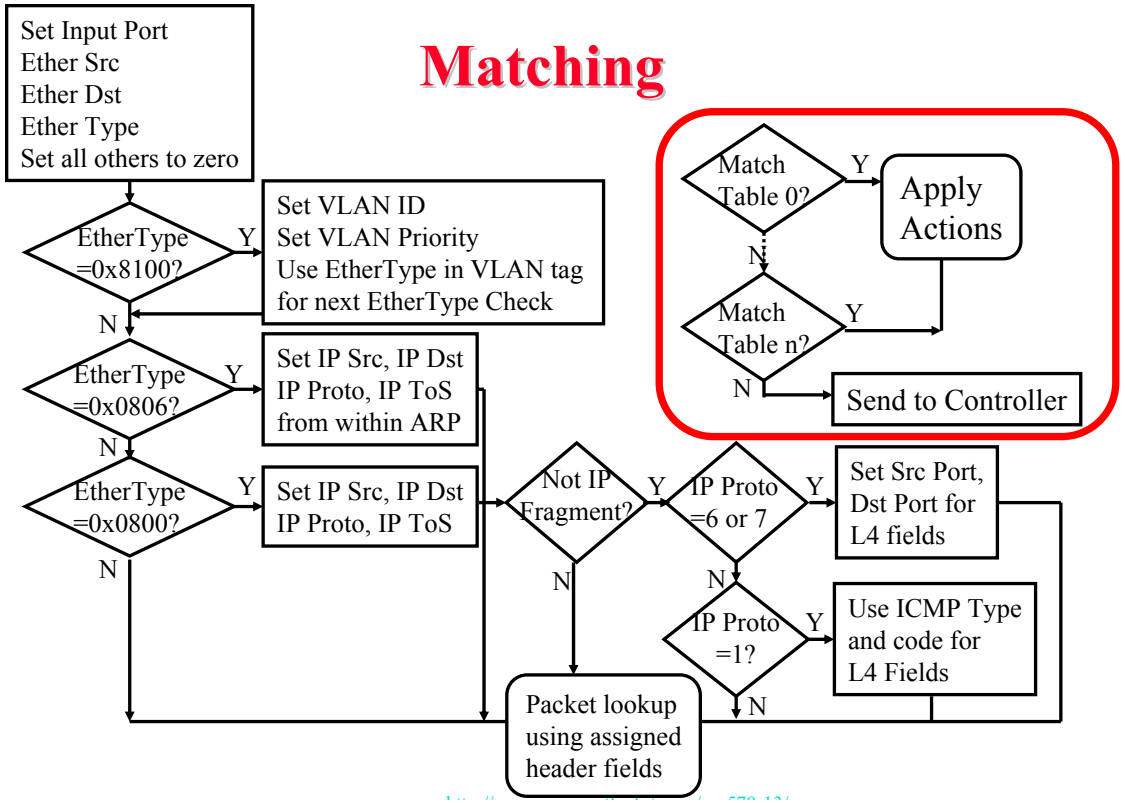
- **All**: to all interfaces except incoming interface
- **Controller**: encapsulate and send to controller
- **Local**: send to its local networking stack
- **Table**: Perform actions in the flow table
- **In_port**: Send back to input port
- **Normal**: Forward using traditional Ethernet
- **Flood**: Send along minimum spanning tree except the incoming interface

Enqueue: To a particular queue in the port → QoS

Drop

Modify Field: E.g., add/remove VLAN tags, ToS bits, Change TTL

Matching Procedure



Counters

Per Table	Per Flow	Per Port	Per Queue
Active Entries	Received Packets	Received Packets	Transmit Packets
Packet Lookups	Received Bytes	Transmitted Packets	Transmit Bytes
Packet Matches	Duration (secs)	Received Bytes	Transmit Overrun Errors
	Duration (nanosecs)	Transmitted Bytes	
		Receive Drops	
		Transmit Drops	
		Receive Errors	
		Transmit Errors	
		Receive Frame Alignment Errors	
		Receive Overrun Errors	
		Receive CRC Errors	
		Collisions	

SDN hands-on

ComNetsEmu

Following hands-on activities will be performed using ComNetsEmu

ComNetsEmu network emulator includes:

- Mininet
- Docker
- Several SDN/NFV examples

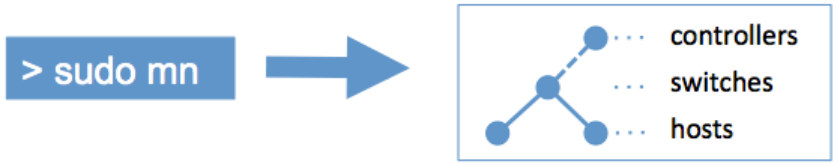
Instructions here:

https://git.comnets.net/public-repo/comnetsemu

Mininet: Overview

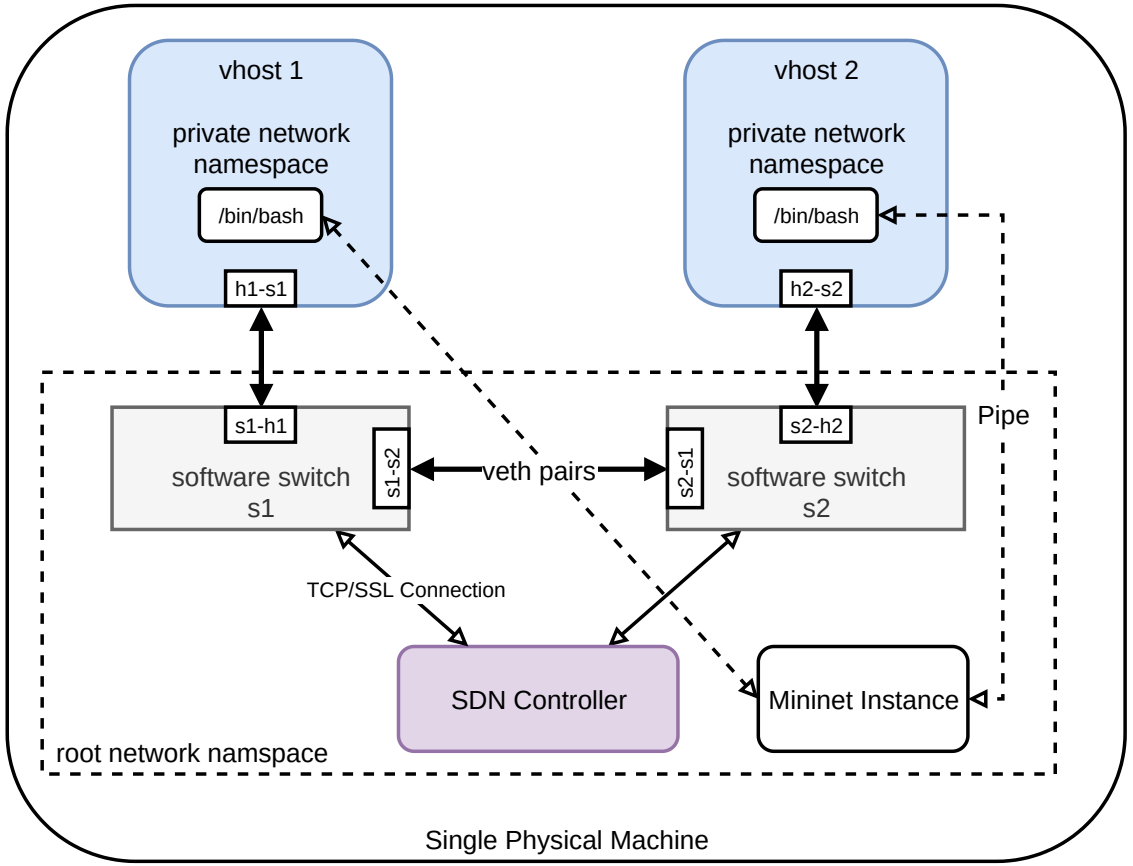
Mininet (mininet.org)

- A network emulator which creates realistic virtual network
- Runs real kernel, switch and application code on a single machine

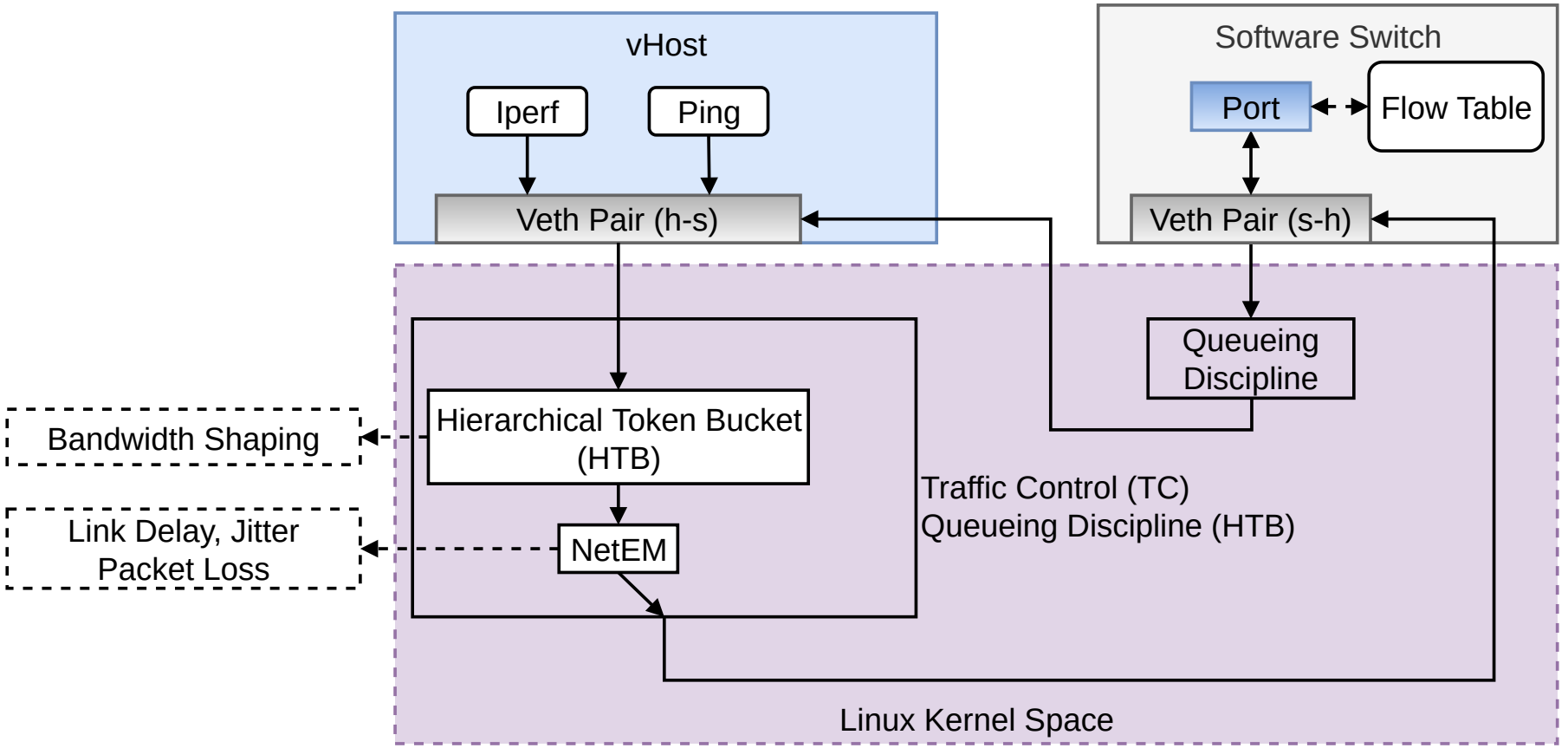


- Provides both Command Line Interface (CLI) and Application Programming Interface (API)
  - CLI: interactive commanding
  - API: automation

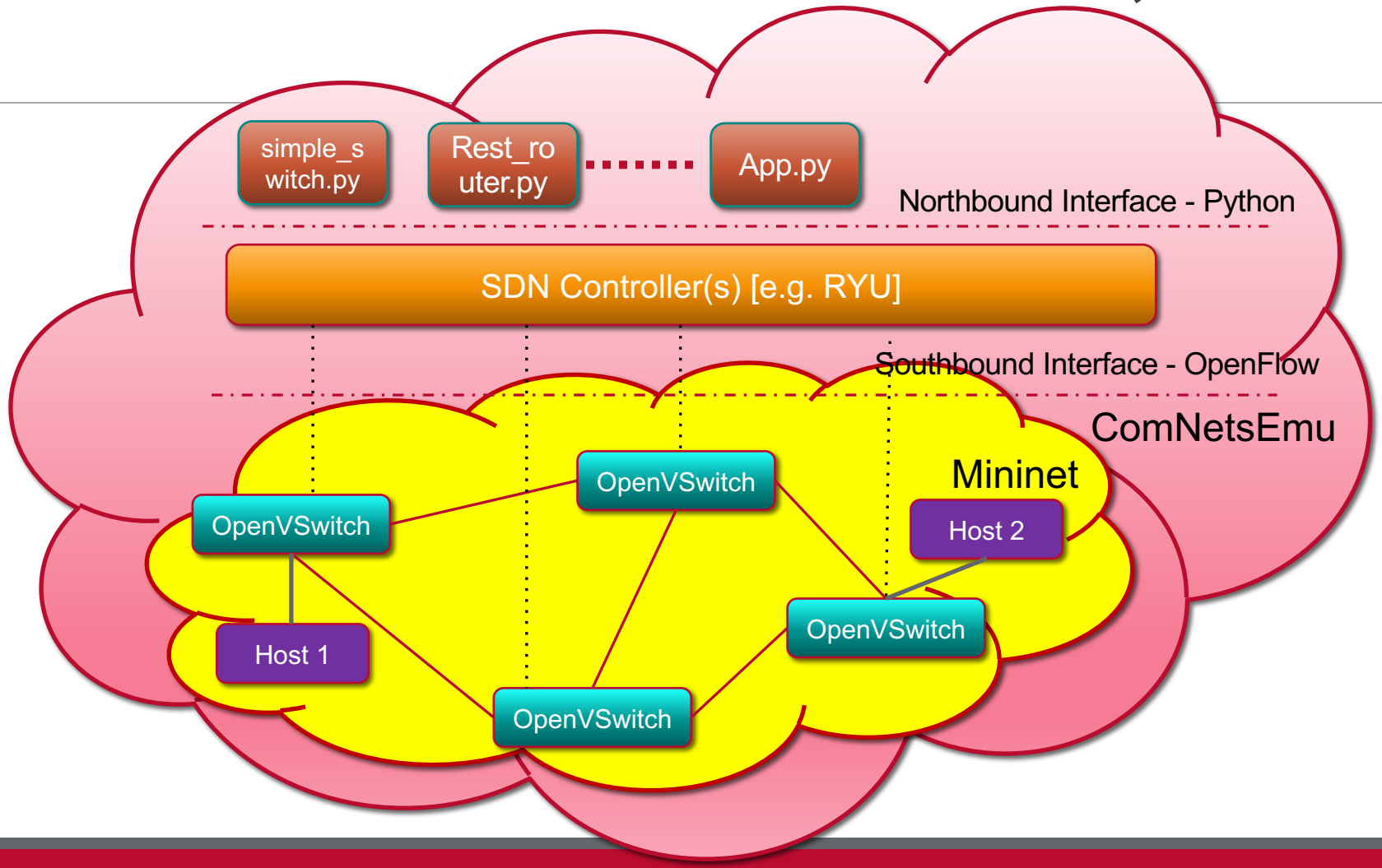
Mininet: Overview



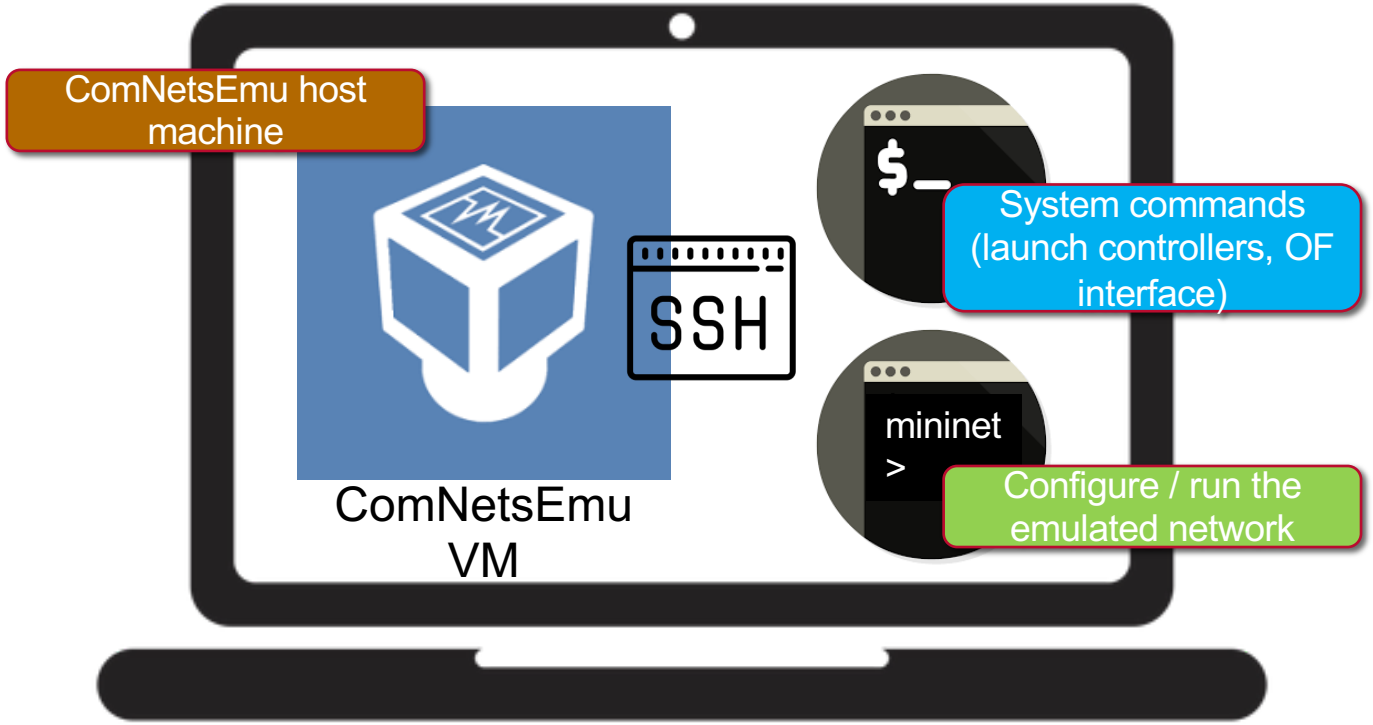
Demystifying Mininet Configurable Data Plane



Hands on SDN architecture, sketch



Hands on Setup



Two Flow Insertion Methods

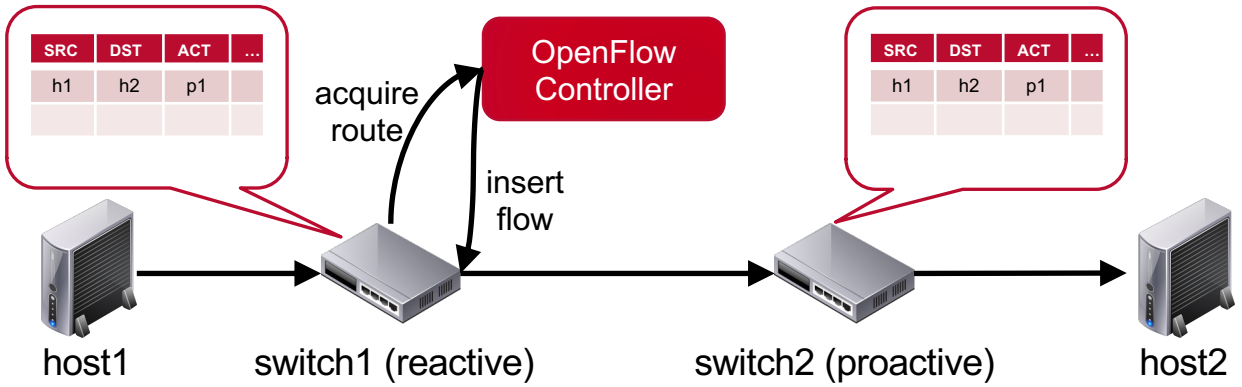
We will experiment with both flow insertion strategies:

Proactive Flow Insertion (e.g. switch 2)

- Flow can be inserted proactively by the controller in switches before packets arrive

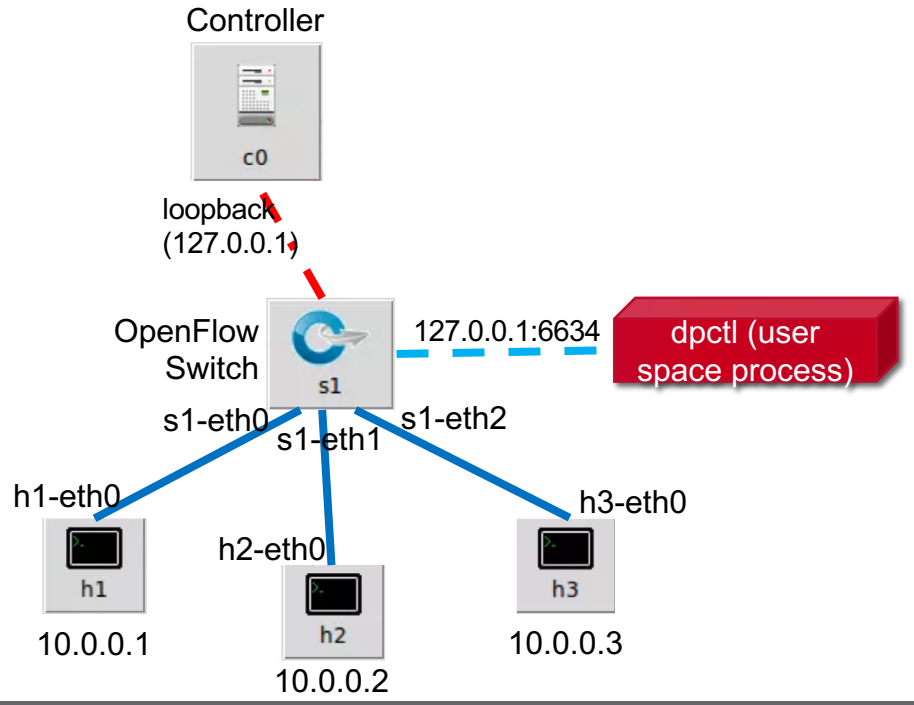
Reactive Flow Insertion (e.g. switch 1)

- A non-matched packet reaches an OpenFlow switch, it is sent to the controller, based on the packet an appropriate flow is inserted



Start a simple network

```
$ sudo mn --topo single,3 --mac --switch ovsk --controller remote
```



What did we do?

```
$ sudo mn --topo single,3 --mac --switch ovsk --controller remote
```

Created 3 virtual hosts, each with a separate IP address.

Created a single OpenFlow software switch in the kernel with 3 ports.

Connected each virtual host to the switch with a virtual ethernet cable.

Set the MAC address of each host equal to its IP.

Configure the OpenFlow switch to connect to a remote controller.

Some relevant mininet commands

To see the list of nodes available, in the Mininet console, run:

```
mininet> nodes
```

To see a list of available commands, in the Mininet console, run:

```
mininet> help
```

To run a single command on a node, prepend the command with the name of the node. For example, to check the IP of a virtual host, in the Mininet console, run:

```
mininet> h1 ifconfig
```

The alternative - better for running interactive commands and watching debug output - is to spawn an xterm for one or more virtual hosts. In the Mininet console, run (might not work on Comnetsemu):

```
mininet> xterm h1 h2
```

If Mininet is not working correctly (or has crashed and needs to be restarted), first quit Mininet if necessary (using the exit command, or control-D), and then try clearing any residual state or processes using:

```
$ sudo mn -c
```

ovs-ofctl example usage

Create another terminal window:

```
$ sudo ovs-ofctl show s1
```

The show command connects to the switch and dumps out its port state and capabilities.

Here's a more useful command:

```
$ sudo ovs-ofctl dump-flows s1
```

Since we haven't started any controller yet, the flow-table should be empty.

Ping test

Now, go back to the mininet console and try to ping h2 from h1. In the Mininet console:

```
mininet> h1 ping -c3 h2
```

Note that the name of host h2 is automatically replaced when running commands in the Mininet console with its IP address (10.0.0.2).

Do you get any replies? Why? Why not?As you saw before, switch flow table is empty.

Besides that, there is no controller connected to the switch and therefore the switch doesn't know what to do with incoming traffic, leading to ping failure.

Ping test – manual config (Proactive)

You'll use ovs-ofctl to manually install the necessary flows (OpenFlow Proactive mode). In your SSH terminal:

```
$ sudo ovs-ofctl add-flow s1 in_port=1,actions=output:2
$ sudo ovs-ofctl add-flow s1 in_port=2,actions=output:1
```

This will forward packets coming at port 1 to port 2 and vice-versa. Verify by checking the flow-table:

```
$ sudo ovs-ofctl dump-flows s1
```

Run the ping command again. In your mininet console:

```
mininet> h1 ping -c3 h2
```

Do you get replies now? Check the flow-table again and look the statistics for each flow entry. Is this what you expected to see based on the ping traffic?

Run Wireshark

The VM image includes the OpenFlow Wireshark dissector pre-installed. Wireshark is extremely useful for watching OpenFlow protocol messages, as well as general debugging.

```
$ sudo wireshark &
```

You'll probably get a warning message for using wireshark with root access. Press OK.

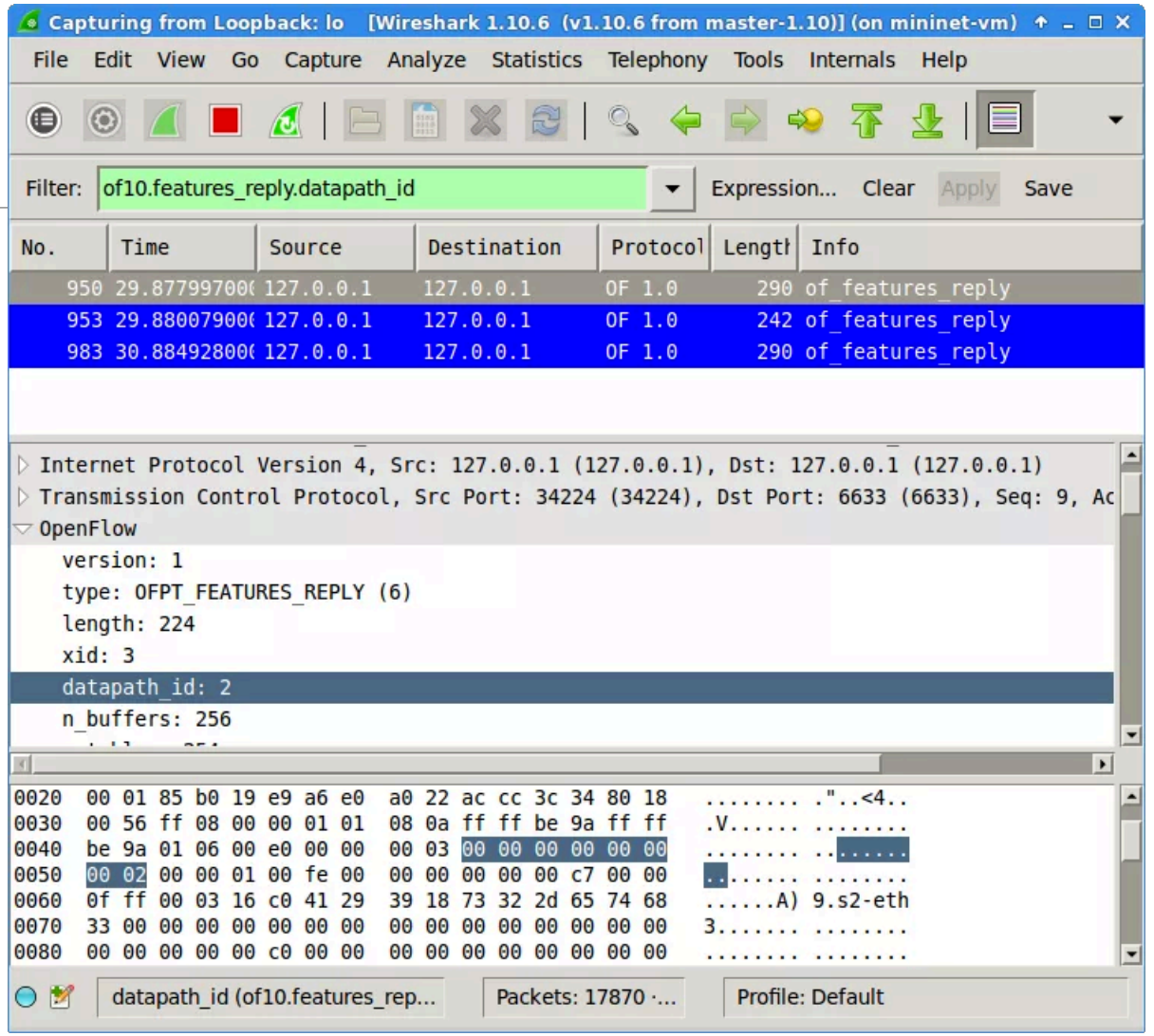
Now, set up a filter for OpenFlow control traffic, by using the 'tcp port 6653' filter (Capture->Options).

Click on Capture->Interfaces in the menu bar. Click on the Start button next to 'lo', the loopback interface. You may see some packets going by.

Press the apply button to apply the filter to all recorded traffic.

(See https://wiki.wireshark.org/OpenFlow for more info on OF support in Wireshark)

Wireshark GUI



Wireshark configuration

Start Wireshark on the local loopback interface and the listening port of the controller (6653)

Note: please use «tcp port 6653» parameter in the capture interface to avoid recording un-necessary packets

You should see a bunch of messages displayed in Wireshark, where the switch is trying to contact the SDN controller

The RYU Controller

Versatile controller included in ComNetsEmu

Programmable in python

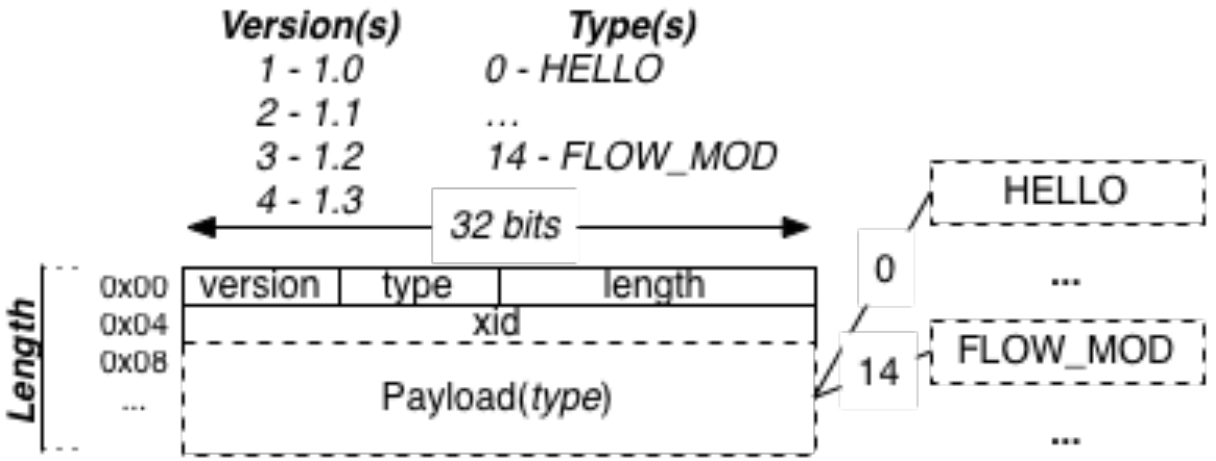
Example:

RYU controller transforming OFSwitches into L2 Learning Switches:

```
$ cd /home/vagrant/comnetsemu_dependencies/ryu-v4.34/ryu/ryu/app
$ ryu-manager simple_switch.py
```

OpenFlow messages

Message format:



See all messages here:

http://flowgrammable.org/sdn/openflow/message-layer/

OpenFlow messages

Hello is used by either controller or switch during connection setup. It is used for version negotiation. When the connection is established, each side must immediately send a Hello message with the version field set to the highest version supported by the sender.

EchoReq in this version has a header and optionally a variable length field, data, containing arbitrary information, such as latency (timestamp), bandwidth, liveness, etc. EchoRes shares the same structure with EchoReq.

The EchoRes data is an exact copy of the data present from the corresponding EchoReq.

OpenFlow messages

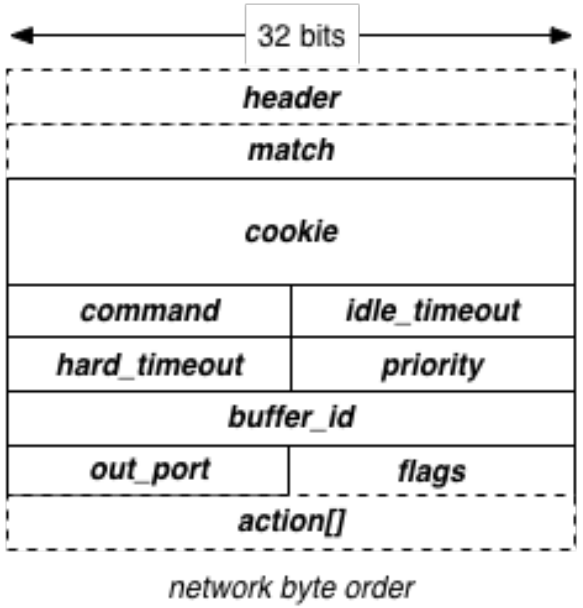
The PacketIn message is a way for the switch to send a captured packet to the controller, e.g. as a result of a match asking for this behavior, or from a miss in the match tables.

The controller has the ability to inject packets into the data plane of a particular switch using the PacketOut message.

OpenFlow messages

FlowMod is one of the main messages, it allows the controller to modify the state of an OpenFlow switch.

This message begins with the standard header and is followed by match, a cookie, which is an opaque field set by the controller, and command which specifies the type of flow table modification.



What about the Northbound interface?

All controllers offer at least a Northbound interface

RYU for example offers a python NB interface

You can find the pre-defined apps here:

```
$ cd /home/vagrant/comnetsemu_dependencies/ryu-v4.34/ryu/ryu/app
```

Feel free to experiment with them, more info here:

https://github.com/osrg/ryu

Exercise: go there and check how to create a hub or a learning switch, or a router (https://osrg.github.io/ryu-book/en/html/rest_router.html)

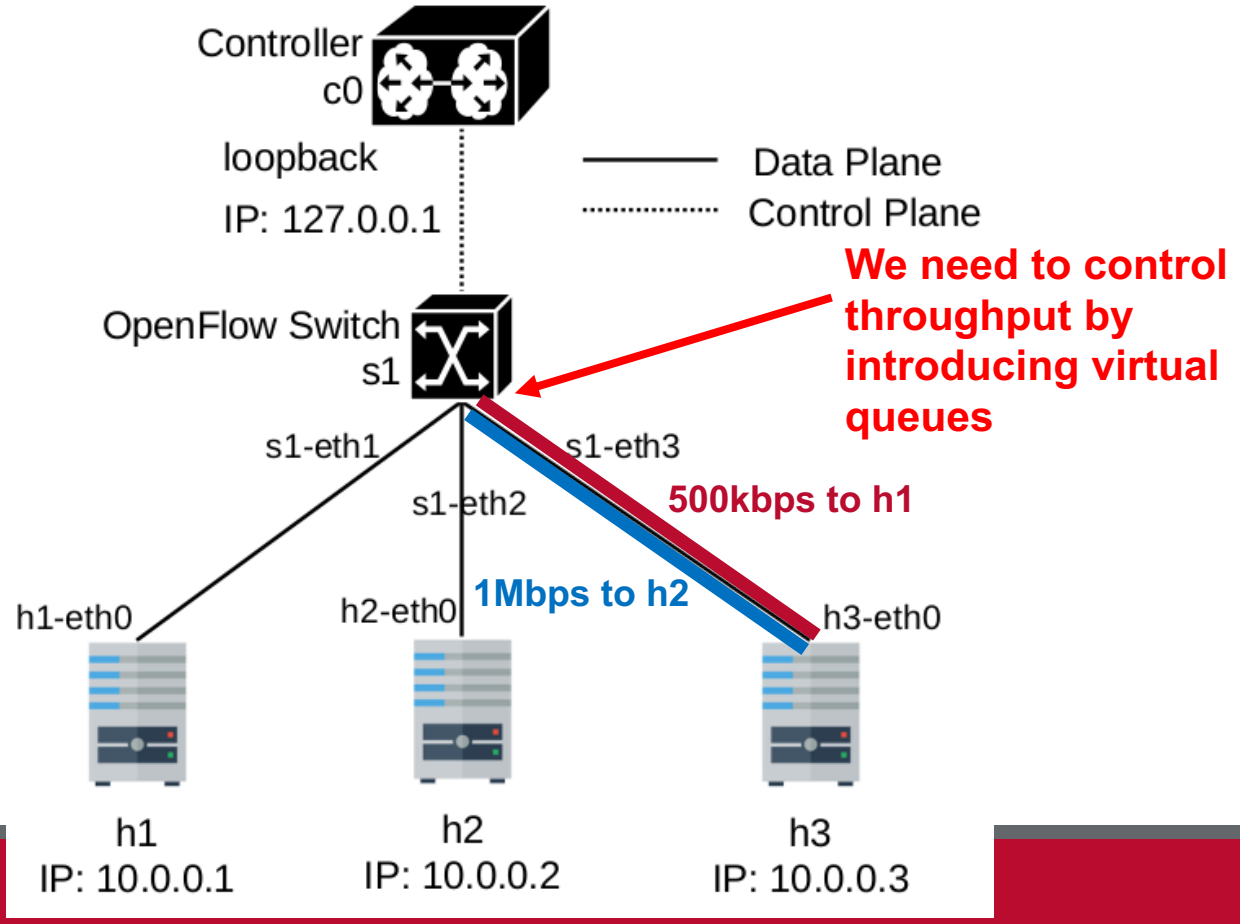
Network Slicing Bulding Blocks

To implement network slicing, we need to define different virtual network topologies, where topology and capacity differ from slice to slice and slices are isolated

To this aim, we need to gain control over the ports and routing strategies of the OF switches as well as over the related scheduling discipline

- To install and control virtual queues in OF Switches

Topology used for link capacity slicing



QoS and Resource Allocation

Generate a simple topology:

```
$ sudo mn --mac --switch ovsk --topo single,3
```

QoS configuration at s1-eth3:

```
$ sudo ovs-vsctl set port s1-eth3 qos=@newqos -- \
--id=@newqos create QoS type=linux-htb \
other-config:max-rate=10000000000 \
queues:123=@1q \
queues:234=@2q -- \
--id=@1q create queue other-config:min-rate=10000 other-config:max-rate=500000 -- \
--id=@2q create queue other-config:min-rate=10000 other-config:max-rate=1000000
```

Check Traffic Control (TC) queues with:

```
$ tc class list dev s1-eth3
```

Queue discipline:

```
$ tc qdisc show
```

QoS and Resource Allocation

Associate flows to virtual queues:

```
#everything on one line only
$ sudo ovs-ofctl add-flow s1
ip,priority=65500,nw_src=10.0.0.1,nw_dst=10.0.0.3,idle_timeout=0,
actions=set_queue:123,normal
#same here
$ sudo ovs-ofctl add-flow s1
ip,priority=65500,nw_src=10.0.0.2,nw_dst=10.0.0.3,idle_timeout=0,
actions=set_queue:234,normal
```

Check that flows are installed:

```
$ sudo ovs-ofctl dump-flows s1
```

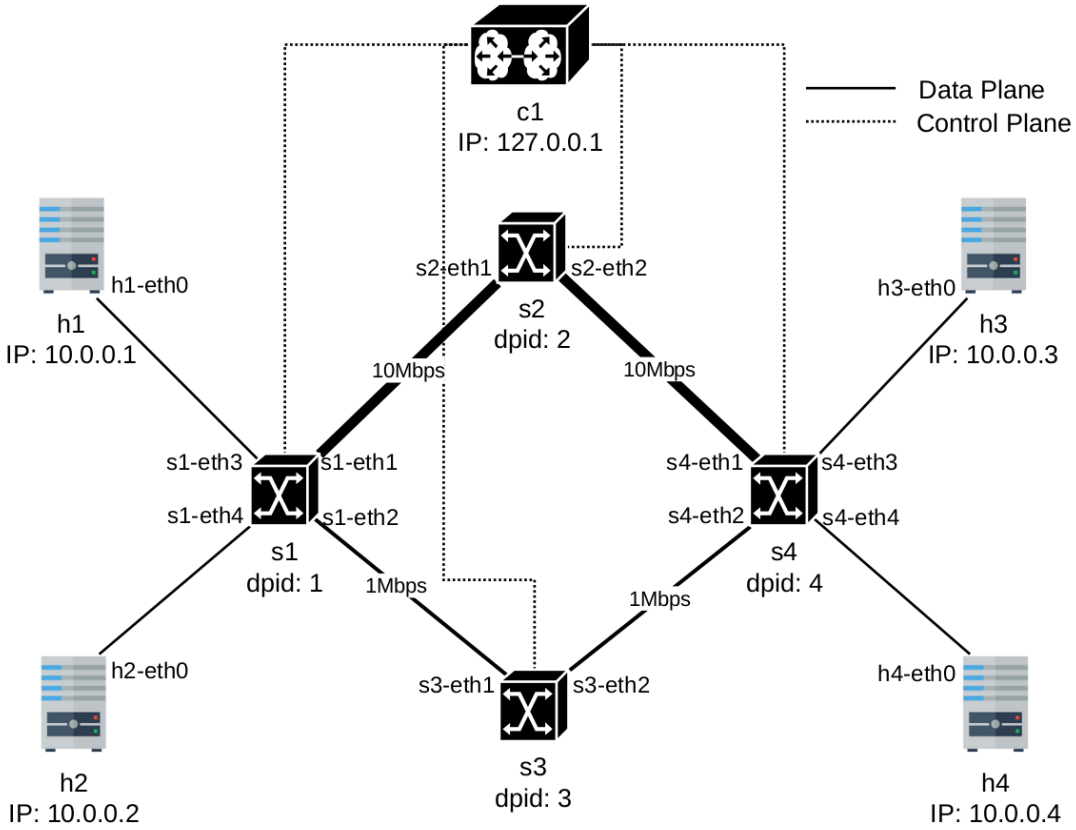
QoS and Resource Allocation

Test bandwidth:

```
mininet> iperf h1 h3
*** Iperf: testing TCP bandwidth between h1 and h3
*** Results: [‘477 kbits/sec’, ‘928 kbits/sec']
```

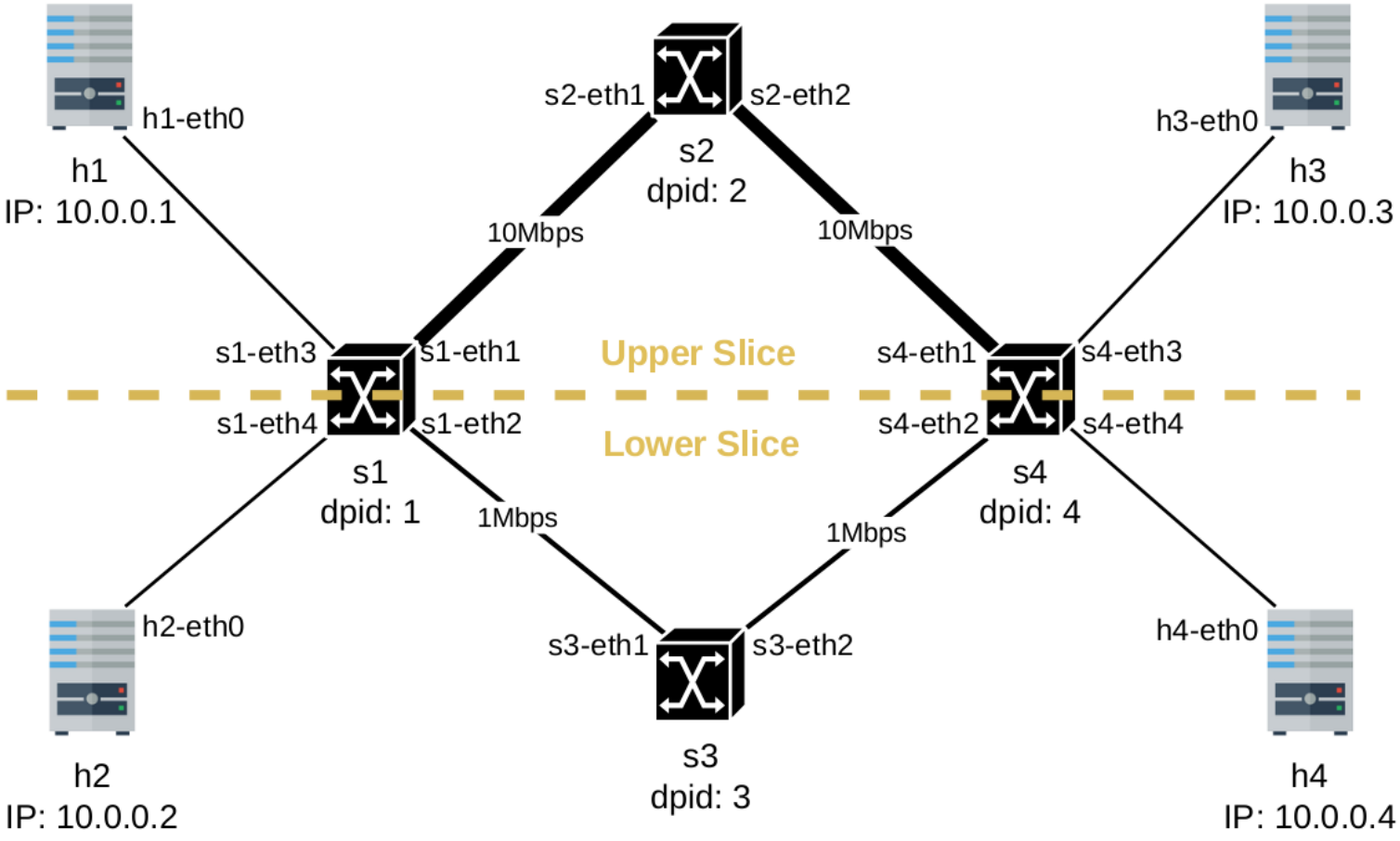
```
mininet> iperf h2 h3
*** Iperf: testing TCP bandwidth between h2 and h3
*** Results: [‘992 kbits/sec’, ‘1.2 Mbits/sec']
```

ComNetsEmu example for network slicing [*]

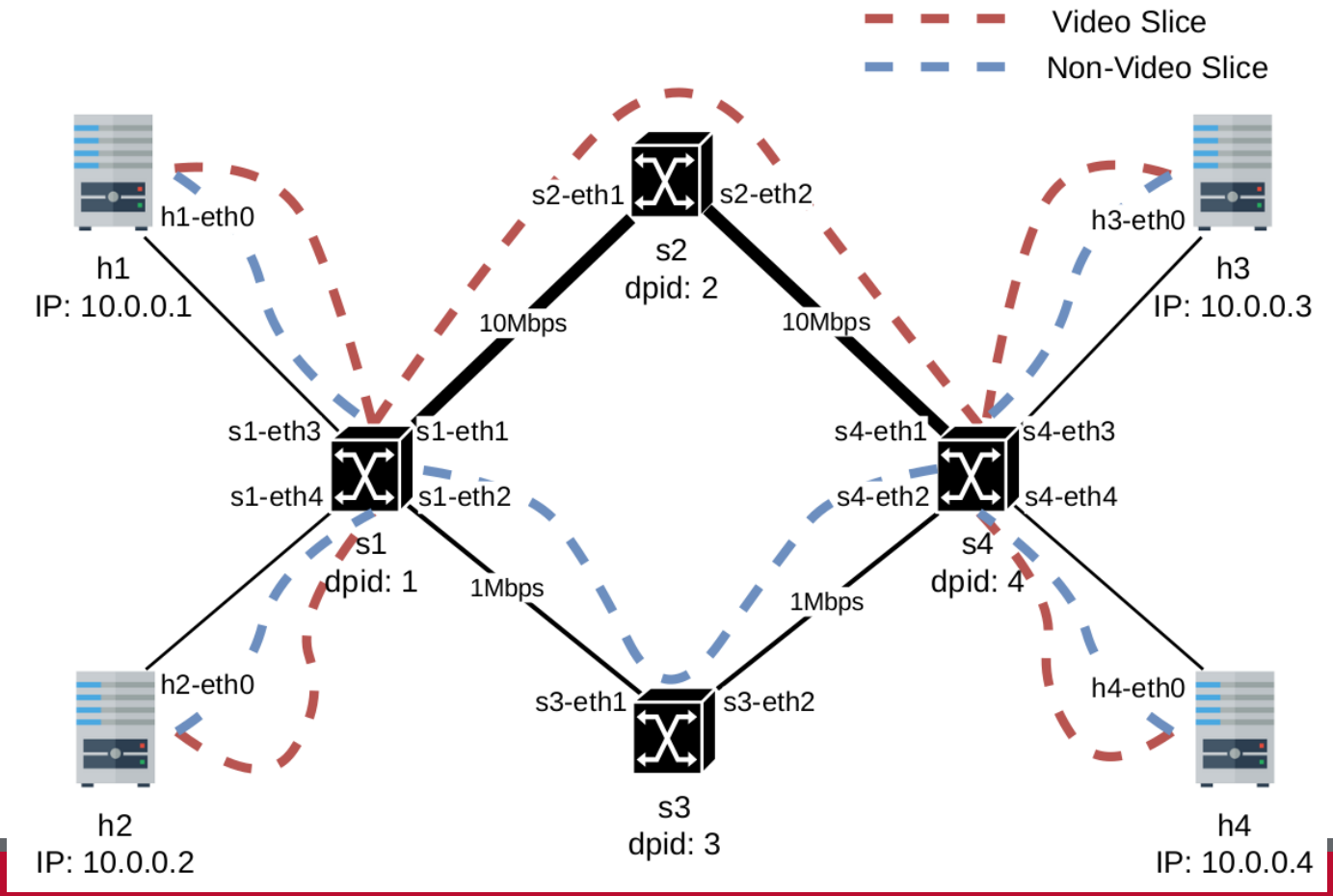


[*] Examples in the following directory:
comnetsemu/app/realizing_network_slicing/
Please read README.md

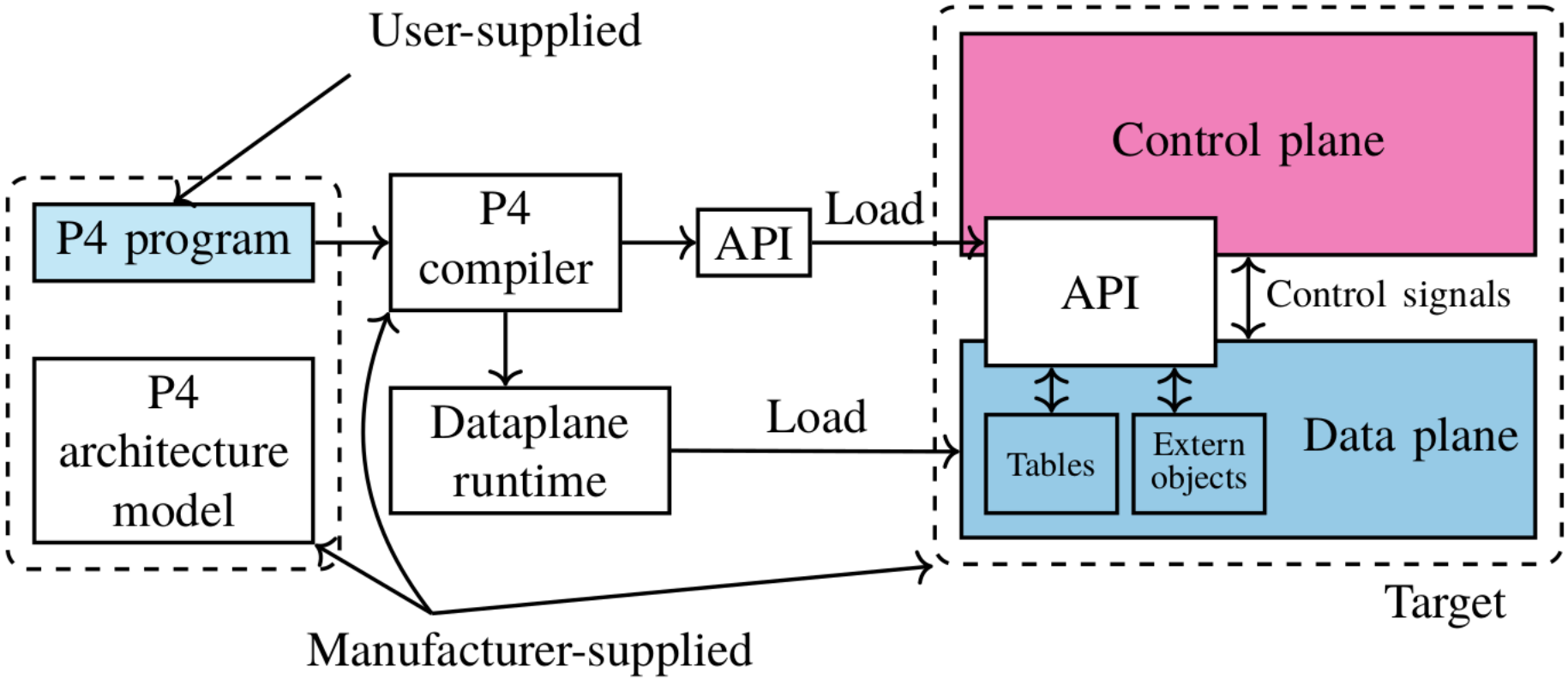
Example 1: topology slicing



Example 2: service slicing



Programming of network devices with P4



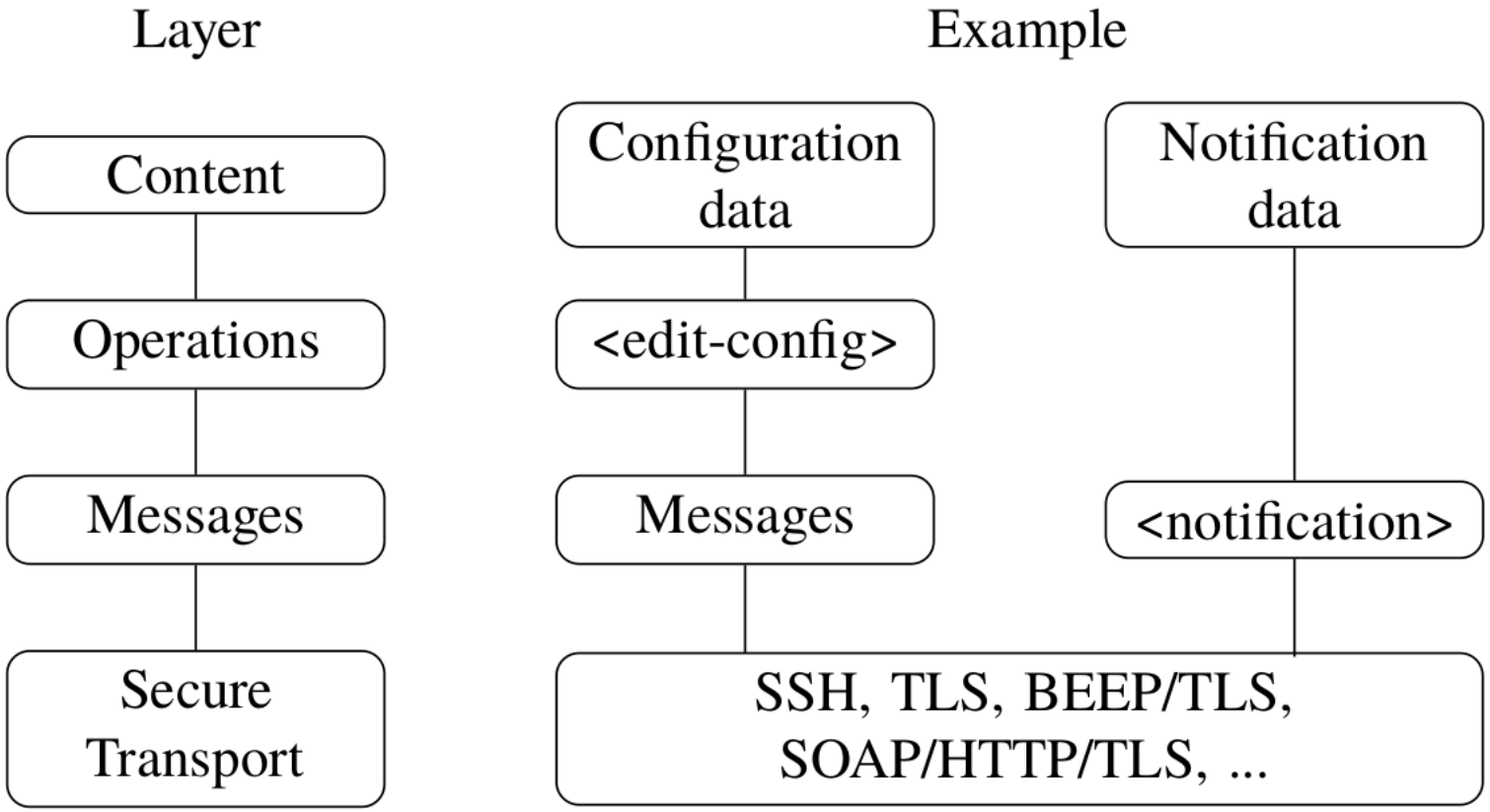
NETCONF

The Network Configuration Protocol (NETCONF) is a network management protocol developed and standardized by the IETF. It was developed in the NETCONF working group and published in December 2006 as RFC 4741[2] and later revised in June 2011 and published as RFC 6241.

NETCONF provides mechanisms to install, manipulate, and delete the configuration of network devices. Its operations are realized on top of a simple Remote Procedure Call (RPC) layer.

The NETCONF protocol uses an Extensible Markup Language (XML) based data encoding for the configuration data as well as the protocol messages. The protocol messages are exchanged on top of a secure transport protocol.

NETCONF protocol layers



https://docs.opendaylight.org/en/stable-oxygen/user-guide/netconf-user-guide.html

NETCONF Tutorial

https://www.youtube.com/watch?feature=oembed&v=xoPZO1N-x38